A Practical Method for Computing Delaunay triangulations in the Euclidean metric

Michael Jünger   Volker Kaibel   Stefan Thienel

Institut für Informatik
Universität zu Köln

June 7, 1994

Abstract

The correctness of many algorithms for computing Delaunay triangulations for the Euclidean metric (as well as for several other problems in Computational Geometry) basically depends on the correct evaluation of the signs of certain arithmetical expressions with integer operands. Since the numbers to deal with often exceed the bounds up to which computers are able to calculate exactly, one has to employ expensive software arithmetic ("big integer packets") to provide correctness in many cases. We present a method to decide dynamically (i.e., for each evaluation occurring during a run of the used algorithm) if it is necessary to perform it by software arithmetic or if one can guarantee the correct evaluation when using a certain "inexact" hardware arithmetic, e.g., the floating point arithmetic of the used system. We apply this method to the computation of Delaunay triangulations and report about some computational experiments.

Keywords: Delaunay triangulation, Voronoi diagram, Computational Geometry, Robust Algorithms

1 Introduction

A *Voronoi diagram* (with respect to the Euclidean metric $d : \mathbb{R}^2 \times \mathbb{R}^2 \longrightarrow \mathbb{R}$) of a set $\Omega = \{g_1, \ldots, g_n\} \subset \mathbb{R}^2$ of n *generators* $g_1, \ldots, g_n$ is the collection of the n polygons

$$P_i := \{p \in \mathbb{R}^2 \mid d(p, g_i) \leq d(p, g_j)\ \forall j \in \{1, \ldots, n\}\}.$$

The "skeleton" formed by the boundaries of these polygons can be viewed as a planar graph, whose dual graph is called a *Delaunay triangulation* of Ω. Because such a De-

launay triangulation contains much information about the neighborhood structure of the generators it is used as a tool in several algorithms.

Many algorithms that compute Delaunay triangulations (via Voronoi diagrams), e.g., the divide-and-conquer method of SHAMOS AND HOEY [9], the sweepline algorithm of FORTUNE [2] and the incremental approaches of GREEN AND SIBSON [3] or OHYA, IRI AND MUROTA [8], use as basic numerical operation the intersection of two lines in the plane. However, even when working with generators having only integer coordinates (as we will presume in the sequel), these intersection points may have rational coordinates that have nonending (periodical) binary representations, and hence cannot be represented correctly as usual floating point numbers. This problem and the possibility of intersection points "far outside", in general, cause errors when computing Delaunay triangulations with implementations of the algorithms mentioned above using floating point arithmetic.

SUGIHARA [11] proposed an algorithm for computing a Delaunay triangulation without intersecting lines. His method needs numerical computations just to evaluate the three functions we will describe next. For any three noncollinear points $a, b, c \in \mathbb{R}^2$ we call the interior of the circle where they all are located on $int(a,b,c)$ and its exterior $ext(a,b,c)$. By $\overrightarrow{ab}$ we denote the directed line between a point $a \in \mathbb{R}^2$ and a point $b \in \mathbb{R}^2$, $b \neq a$. The three functions are:

`Closer`(z,a,b) :	TRUE	if $d(z,a) < d(z,b)$
	FALSE	otherwise
`CheckVP`(z,a,b,c) :	IN	if $z \in int(a,b,c)$
	OUT	if $z \in ext(a,b,c)$
	ON	otherwise
`OnLeftSide`(z,a,b) :	TRUE	if z is on the left side of $\overrightarrow{ab}$
	FALSE	otherwise

Proposition 1.1 shows the formulas given in JÜNGER, REINELT AND ZEPF [5] to evaluate these functions. We denote the x-coordinate of a point $p \in \mathbb{R}^2$ by p_x, its y-coordinate by p_y.

Proposition 1.1 *Let $a, b, c, z \in \mathbb{R}^2$. Then*

(i) `Closer`$(z,a,b) = TRUE \iff$

$$\begin{aligned} 0 \;>\; & 2(b_x - a_x)(z_x - b_x) + (b_x - a_x)(b_x - a_x) \\ & +2(b_y - a_y)(z_y - b_y) + (b_y - a_y)(b_y - a_y) \end{aligned}$$

(ii) `CheckVP`$(z,a,b,c) = \left\{ \begin{array}{l} IN \\ OUT \\ ON \end{array} \right\} \iff \frac{P}{Q} \left\{ \begin{array}{l} < 0 \\ > 0 \\ = 0 \end{array} \right\}$, *where*

$$\begin{aligned}
P &= (z_x - a_x)(b_x - a_x)(z_x - b_x)(c_y - a_y) \\
&\quad -(z_x - a_x)(c_x - a_x)(z_x - c_x)(b_y - a_y) \\
&\quad -(c_x - b_x)(b_x - a_x)(c_x - a_x)(z_y - a_y) \\
&\quad +(z_y - a_y)(c_y - a_y)(z_y - c_y)(b_x - a_x) \\
&\quad -(z_x - a_y)(a_y - b_y)(b_y - a_y)(c_x - a_x) \\
&\quad +(c_y - a_y)(c_y - b_y)(b_y - a_y)(z_x - a_x),
\end{aligned}$$

$$Q = (c_x - a_x)(b_y - a_y) - (b_x - a_x)(c_y - a_y)$$

(iii) `OnLeftSide`$(z, a, b) = TRUE \iff$

$$0 > (b_y - a_y)(z_x - a_x) - (z_y - a_y)(b_x - a_x)$$

Using these formulas, one has not to perform any divisions, and hence (recall that we assume integer generators) just has to deal with integers during the computations. However, the (intermediate) results of the calculations required by the formulas of Proposition 1.1 may become rather large and cause overflows when using standard integer arithmetic.

For generators lying on a grid $\{0, 1, \ldots, A\} \times \{0, 1, \ldots, A\}$ Jünger, Reinelt and Zepf [5] give the following guarantee for a b-bit 2-complement integer arithmetic.

Proposition 1.2 *No numerical errors occur in the calculations required by the formulas of Proposition 1.1 as long as*

$$A \leq \left\{ \begin{array}{lll} 8 & \textit{if} & b = 16 \\ 137 & \textit{if} & b = 32 \\ 35211 & \textit{if} & b = 64 \end{array} \right\}$$

holds.

But, considering the fact that on many current computers (e.g., on the SPARCstations of SUN where our implementation runs) the integer arithmetic works with a word length of 32 bits, the consequence of Proposition 1.2 is the restriction of the grid size to $A \leq 137$. Even on systems with an integer arithmetic of word length 64 bits coordinates of the generators must not exceed 35211. Of course, many real world problems have distinctively larger coordinates after scaling the (usually rational) coordinates of the generators to integer numbers (e.g., the problems in the TSPLIB of Reinelt [10]).

There seem to be two ways out:

1. Instead of using the integer arithmetic of the system, it is possible to perform the calculations by the floating point arithmetic of the computer. Normally, this arithmetic permits the representation of sufficiently large numbers, but it becomes

inaccurate at a certain size of the numbers, which depends on the length of fraction in the floating point representation. Consequently, one is able to run the algorithm with larger coordinates this way, however without a guarantee of correctness (and even without a guarantee of termination, when using an incremental algorithm that is based on the fact that the current Delaunay triangulation is a correct one at each incremental step).

2. It is possible to make the word length for integer calculations as large as desired (limited only by the storage capacity of the machine) by using a software-arithmetic, which represents integer numbers in blocks of words of the integer arithmetic of the used system. Hence, the algorithm works correctly with arbitrarily large coordinates.

To be able to guarantuee by Proposition 1.2 that the algorithm produces the expected result, it is necessary to employ (comparatively slow) software arithmetic from a certain (comparatively small) size of the coordinates on. Nevertheless, it is not necessary (even in cases of very large coordinates) to perform every calculation by software arithmetic, because Proposition 1.2 is a worst case estimation. For that reason, our approach is to decide for every call of one of the three functions `Closer`, `CheckVP` or `OnLeftSide` by examining the parameters $z, a, b(, c)$, if the required computation can be performed correctly by (floating point) hardware arithmetic, or if the use of software arithmetic is necessary. The main goal of this paper is to present our criterion for this decision, which we will do in the following section. The third section will give some of our computational results.

Independently, FORTUNE AND VAN WYCK [1] developed a similar approach. They propose a general framework for the exact evaluations of "primitives for geometric algorithms" and applied it to the computation of Delaunay triangulations.

2 The routine sign_of_expression

Our method was motivated by the following observation. When using floating point arithmetic, our implementation of an incremental algorithm based on OHYA, IRI AND MUROTA [8] and SUGIHARA [11] computes the correct results for nearly all instances, even if the coordinates of the generators are very large. The reason is that floating point arithmetic gives the possibility to perform integer computations (as addition, subtraction and multiplication) with very large (intermediate) results and with relatively small errors in comparison to the big errors that occur in case of an overflow in integer arithmetic. The careful estimation of those "small" errors, together with the fact that according to Proposition 1.1 we are only interested in the signs of the values of some numerical expressions, will give us a very effective criterion that requires only a few calls of the software arithmetic even for instances with very large coordinates.

The idea of our technique is the following: According to Proposition 1.1 we have to

evaluate expressions of the form

$$\sum_{1\le i\le r}\left(s_i\prod_{1\le j\le\gamma_i}t_{ij}\right),$$

where t_{ij} are certain coordinate differences and $s_i\in\{-1,+1\}$. Let P_{ex} be the exact sum of the positive summands in the expression above and N_{ex} the exact sum of the negative summands. First of all we compute all the summands (i.e., the products $s_i\otimes(\otimes_{1\le j\le\gamma_i}t_{ij})$, where $\otimes$ is the floating point multiplication on the machine) and after that we add up the positive ones to obtain a value P, and the negative ones to obtain a value N. The sign of the expression which we have to evaluate only depends on the relation of the sizes of P_{ex} and $|N_{ex}|$. In the sequel we will give an estimation for the error $|P-P_{ex}|$ (resp. $||N|-|N_{ex}||$) which one can obtain easily from P (resp. N). Having an estimation of these errors one can deduce a criterion that guarantees that P and $|N|$ have the same $(<,=,>)$-relation as P_{ex} and $|N_{ex}|$.

Our approach does not directly access to floating point representations. It only requires a (hardware) arithmetic which has some features that are especially satisfied by the IEEE STANDARD [4] (cf. Proposition 2.2). But one can use our method also with any other arithmetic which shares these features. By abuse of notation we call the used arithmetic "floating point arithmetic" anyway. We formulate these features mentioned above in the following

Conventions and Assumptions:

Let

$$M,L\in\mathbb{N}.$$

M will become the storing size of the fraction and L the value of the maximal exponent of a floating point number. Moreover, let

$$D\subset\{-2^L,\dots,2^L\}$$

be a set of integer numbers. In our case it will be the set of integer numbers which are representable in the floating point arithmetic. Then D can be a proper subset of $\{-2^L,\dots,2^L\}$, because mostly $M<L$ holds, so that the lower digits of large integer numbers cannot be stored. The numbers in D then have the following form:

$$\underbrace{\underbrace{*\dots*}_{M}0\dots0}_{\le L},$$

where the stars stand for digits in $\{0,1\}$.

We define for all integer numbers $\alpha\in\mathbb{Z}$

$$l(\alpha):=\left\{\begin{array}{cl}\lfloor\log(|\alpha|)\rfloor+1 & \text{if }\alpha\neq 0\\ 1 & \text{otherwise}\end{array}\right\},$$

where log denotes the binary logarithm. $l(\alpha)$ is the binary length of the number $|\alpha|$.

For $a, b \in \mathbf{D}$ let

$$a + b,\ a - b,\ a * b\ (= ab)$$

be the sum, the difference and the product in the ring of integers, and

$$a \oplus b,\ a \ominus b,\ a \otimes b$$

the results of addition, subtraction and multiplication (resp. the operations themselves) in the floating point arithmetic.

As we have already suggested we want to use the feature of floating point arithmetic that the operations addition, subtraction and multiplication are performed "nearly correctly" with integer operands. Of course this is only true as long as the result is not bigger than any number representable in the floating point format. To describe this case we define the notion of "overflow" in the following way:

Definition 2.1 *For $a, b \in \mathbf{D}$, $\bullet \in \{+, -, *\}$ and the corresponding machine operation $\odot \in \{\oplus, \ominus, \otimes\}$ the operation $a \odot b$ causes an overflow, if*

$$l(a \bullet b) > L.$$

We make the following

Assumption (#):

1. For $a, b \in \mathbf{D}$ and $\odot \in \{\oplus, \ominus, \otimes\}$,

   $$a \odot b \in \mathbf{D}$$

   should hold, as long as no overflow occurs during the operation.

   This assumption guarantees that the set D is closed under the operations we need. Addition, subtraction or multiplication of two numbers from D should not have a fractional result in the floating point arithmetic.

2. For $a, b \in \mathbf{D}$, $\bullet \in \{+, -, *\}$ and the corresponding machine operation $\odot \in \{\oplus, \ominus, \otimes\}$

   $$|(a \bullet b) - (a \odot b)| \leq 2^{l(a \odot b) - M}.$$

   should hold, as long as no overflow occurs during the operation.

   This assumption quantifies the size of the error for the required operations. If

   $$\underbrace{\underbrace{* \ldots *}_{M} 0 \ldots 0}_{\leq L}$$

   is the result of such an operation the estimation above means that at least the first $M - 1$ digits are computed correctly.

3. For $a, b \in \mathbf{D}$, $a, b \geq 0$ or $a, b \leq 0$

$$l(a \oplus b) \geq l(a),$$

should hold, as long as no overflow occurs during the operation.

We will examine the accumulation of errors in a sequence of operations. This part of the assumption allows us to estimate the length of intermediate results of sequential additions of numbers with the same sign by the length of the total result.

4. For $a, b \in \mathbf{D}$

$$l(a \otimes b) \geq l(a) + l(b) - 1$$

should hold, as long as no overflow occurs during the computation.

This is the analogous assumption to part 3 for multiplication.

5. The system should perform comparisons $(<, =, >, \leq, \geq)$ of numbers $a, b \in \mathbf{D}$ correctly in any case.

6. There should be a function abs, so that for all $a \in \mathbf{D}$

$$abs(a) = |a|$$

holds.

7. For $a, b \in \mathbf{D}$ with $a, b \geq 0$ or $a, b \leq 0$

$$|a \oplus b| = |a| \oplus |b|$$

should hold as long as no overflow occurs during the computation.

This and the next part of the assumption express that addition (of operands with equal sign) and multiplication are performed as if the absolute value of the result would be determined first and then the sign of the result.

8. For $a, b \in \mathbf{D}$

$$|a \otimes b| = |a| \otimes |b|,$$

should hold, as long as no overflow occurs during the computation.

9. Finally we require

$$M > 8.$$

If α is the result of an operation in part 2 with $l(\alpha) \leq 8$ this part of the assumption allows us in connection with part 1 to conclude the correctness of the result.

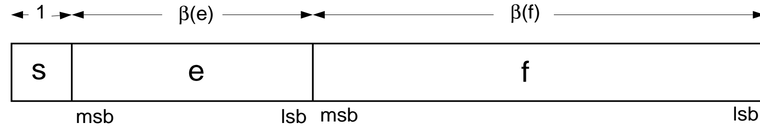
For $t_1, \ldots, t_n \in \mathbf{D}$ and the machine operation $\odot \in \{\oplus, \otimes\}$ we define

$$\odot_{1 \leq i \leq n} t_i := (\ldots((t_1 \odot t_2) \odot t_3) \ldots) \odot t_n,$$

as long as no overflow occurs during the sequence of operations (because of part 1 of the Assumption (#) the expression above is well defined in this case), otherwise we say that $\odot_{1 \leq i \leq n} t_i$ causes an overflow.

Proposition 2.2 *Assumption (#) is satisfied by systems with a floating point arithmetic according to the* IEEE STANDARD *[4].*

Proof: The IEEE STANDARD [4] specifies the following format for floating point numbers (where *msb* and *lsb* stand for the most (resp. least) significant bit):



The fields e and f are bit strings of length $\beta(e)$ resp. $\beta(f)$. We denote with E the (nonnegative) integer number represented in binary format by the string e. So

$$\text{MaxE} := 2^{\beta(e)} - 1$$

is the maximal value of E. Furthermore let

$$\text{bias} := 2^{\beta(e)-1} - 1.$$

Let v be a number represented in floating point format by a triple (s, e, f). We are interested in two cases:

1. $\text{bias} \leq E < \text{MaxE}$
   Then $v = (-1)^s 2^{E-\text{bias}} (1.f)_2$ holds.
2. $E = 0, f = 0 \dots 0$
   Then we have $v = 0$.

In all other cases v is not an integer number.
We set

$$M := \beta(f) + 1 \quad \mathit{and}, \quad L := \text{MaxE} - \text{bias}.$$

Finally let **D** be the set of representable integer numbers.
The IEEE STANDARD [4] requires the following features for the operations $\oplus, \ominus$ und $\otimes$:

> "(...) every operation (...) shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result (...)" (p. 10)

As default rounding mode *Round to Nearest* is prescribed:

> "(...) the representable value nearest to the infinitely precise result shall be delivered; if the two nearest representable values are equally near, the one with its least significant bit zero shall be delivered." (p. 10)

We now check all parts of the assumption (#):

Part 1: Let $a, b \in \mathbf{D}$, $\odot \in \{\oplus, \ominus, \otimes\}$ and $\bullet \in \{+, -, *\}$ be the corresponding ring operation. We define

$$z := a \odot b$$

and assume that no overflow occures during the operation. Then z is the number obtained by *Round to Nearest* from the integer number

$$z' := a \bullet b.$$

Assume $z \notin \mathbf{D}$, then we have $z \in \mathbb{Q} \setminus \mathbb{Z}$. Because z is representable it is guaranteed that

$$\lfloor z \rfloor, \lceil z \rceil \in \mathbf{D}.$$

Since we have $z' \in \mathbb{Z}$ either $\lfloor z \rfloor$ or $\lceil z \rceil$ is closer to z' than z itself, which is a contradiction to the rule *Round to Nearest*.

Part 2: Let be $z' \in \mathbb{Z}$, $l(z') \leq L$ and $z \in \mathbf{D}$ the number obtained from z' by *Round to Nearest*. We have to show that

$$|z' - z| \leq 2^{l(z)-M}.$$

If $l(z') \leq M$ then the assertion is obvious. Otherwise the number z' (omitting the sign) has a binary representation of the form

$$\underbrace{\underbrace{1 * \ldots *}_{M} * \ldots *}_{l(z')}.$$

So we have

$$z' = \alpha 2^{l(z')-M} + \beta,$$

whith some (unique) $\alpha, \beta \in \mathbb{N}$, $2^{M-1} \leq \alpha < 2^M$, $0 \leq \beta < 2^{l(z')-M}$. The rule *Round to Nearest* gives us

$$z = \alpha 2^{l(z')-M} \quad \text{or} \quad z = (\alpha + 1) 2^{l(z')-M}.$$

Since $\alpha \geq 2^{M-1}$, one readily deduces

$$l(z) \geq l(z') - 1$$

from this, an estimation which we need in the sequel.

Case 1: $z = \alpha 2^{l(z')-M}$

In this case $\beta \leq 2^{l(z')-M-1}$ holds, so we can estimate:

$$\begin{aligned} |z' - z| &= z' - z \\ &= (\alpha 2^{l(z')-M} + \beta) - \alpha 2^{l(z')-M} \\ &= \beta \\ &\leq 2^{l(z')-M-1} \\ &\leq 2^{l(z)-M} \end{aligned}$$

Case 2: $z = (\alpha + 1)2^{l(z')-M}$
In this case $\beta \geq 2^{l(z')-M-1}$ holds, so we can estimate:

$$\begin{aligned} |z' - z| &= z - z' \\ &= (\alpha + 1)2^{l(z')-M} - (\alpha 2^{l(z')-M} + \beta) \\ &= 2^{l(z')-M} - \beta \\ &\leq 2^{l(z')-M} - 2^{l(z')-M-1} \\ &= 2^{l(z')-M-1} \\ &\leq 2^{l(z)-M} \end{aligned}$$

Parts 3 and 4: They are obvious because rounding a number by the above rule can never make it shorter (compare also the proof of part 2).

Part 5: The IEEE STANDARD [4] prescribes:

> "Comparisons are correct (...)" (p. 12)

Part 6: Trivial, because only the bit of the sign must be set.

Parts 7 and 8: These two parts follow from the observation that a number a is rounded to A if and only if $-a$ is rounded to $-A$ when using *Round to Nearest*.

Part 9: All formats specified by IEEE STANDARD have $M > 8$. □

Remark 2.3 *The* IEEE STANDARD *[4] especially specifies the formats*

1. *SINGLE-PRECISION:* $\beta(e) = 8, \beta(f) = 23$. *We choose* $M := 24$ *and* $L := 128$.
2. *DOUBLE-PRECISION:* $\beta(e) = 11, \beta(f) = 52$. *We choose* $M := 53$ *and* $L := 1024$.

Our implementation uses the DOUBLE-PRECISION format.

Now we want to examine how errors accumulate in a sequence of additions and multiplications on the machine in order to obtain the mentioned estimation of $|P - P_{ex}|$ and $||N| - |N_{ex}||$ finally. Our approach is as follows: The first goal is to obtain an estimation of the error occurring at the successive addition of some numbers with the same sign. The result will be Proposition 2.5. After that we give an analogous estimation for the multiplication in Proposition 2.6, and in Proposition 2.9 we present the central estimation of the error

$$\left| \sum \prod t_{ij} - \oplus \otimes t_{ij} \right|, \quad t_{ij} \in \mathbf{D}.$$

Proposition 2.11 finally gives us the criterion directly derived from Proposition 2.9 which we will use to decide if there could have been an error in the floating point calculation of the sign of one of the expressions above.

First of all we need a lemma that allows us to apply part 2 of Assumption (#) repeatedly if several numbers with same sign are added.

Lemma 2.4 *Let $a, b \in \mathbf{D}$ and $a, b \geq (\leq)\ 0$. Then we have*

$$a \oplus b \geq (\leq)\ 0,$$

if no overflow occurs during the operation.

Proof: Otherwise, one obtains

$$|a \oplus b| \leq |(a \oplus b) - (a + b)| \leq 2^{l(a \oplus b) - M},$$

from part 2 of Assumption (#), which is a contradiction since $M > 8$. □

Now, we can estimate the error occurring when adding several numbers of the same sign. For the sequel, we assume $n > 1$.

Proposition 2.5 *Let $t_1, \ldots, t_n \in \mathbf{D}$, either all nonnegative or all nonpositive, and let $\alpha := \oplus_{1 \leq i \leq n} t_i$. If no overflow occurs during the computation of α*

$$\Big| \sum_{1 \leq i \leq n} t_i - \alpha \Big| \leq 2^{l(\alpha) - M + (n-2)}$$

holds.

Proof: For $3 \leq j \leq n$ define

$$T_2 := t_1 \oplus t_2 \quad \text{and} \quad T_j := T_{j-1} \oplus t_j$$

(these T_j are the intermediate results during the computation), as well as for $2 \leq j \leq n$

$$F_j := \Big(\sum_{1 \leq i \leq j} t_i \Big) - T_j.$$

So F_j is the error accumulated during the first j additions. Note that we assume no overflows ocurring when computing the T_j. For $3 \leq j \leq n$ we have:

$$\begin{aligned} |F_j| &= \Big| \underbrace{\Big(\sum_{1 \leq i \leq j-1} t_i \Big)}_{F_{j-1} + T_{j-1}} + t_j - (T_{j-1} \oplus t_j) \Big| \\ &\leq |F_{j-1}| + |(T_{j-1} + t_j) - \underbrace{(T_{j-1} \oplus t_j)}_{T_j}| \\ &\leq |F_{j-1}| + 2^{l(T_j) - M}, \end{aligned}$$

where we used part 2 of Assumption (#) for the last inequality.

By induction on $j \in \{2, \ldots, n\}$ we show

$$|F_j| \leq 2^{l(T_j) - M + j - 2},$$

which proves the assertion for $j = n$. In this induction we use that by Lemma 2.4 and part 3 of Assumption (#) we inductively get

$$l(T_j) \geq l(T_{j-1}) \quad \text{for } 2 \leq j \leq n.$$

$j = 2$: part 2 of Assumption (#).
$j > 2$:

$$\begin{aligned} |F_j| &\leq |F_{j-1}| + 2^{l(T_j)-M} \\ &\leq 2^{\overbrace{l(T_{j-1})}^{\leq l(T_j)} - M + (j-1) - 2} + 2^{l(T_j)-M} \\ &\leq 2^{l(T_j)-M+j-3} + 2^{l(T_j)-M+\overbrace{j-3}^{\geq 0}} \\ &= 2^{l(T_j)-M+j-2} \end{aligned}$$

□

We now give an analogous estimation for the multiplication of several numbers.

Proposition 2.6 *Let $t_1, \ldots, t_n \in \mathbf{D}$, $\alpha := \otimes_{1 \leq i \leq n} t_i$. If no overflow occurs during the computation of α we have*

$$\left| \prod_{1 \leq i \leq n} t_i - \alpha \right| \leq 2^{l(\alpha)-M+2(n-2)}.$$

Proof: We define

$$T_2 := t_1 \otimes t_2 \quad \text{and} \quad T_j := T_{j-1} \otimes t_j \quad \text{for} \quad 3 \leq j \leq n,$$

as well as

$$F_j := \Big(\prod_{1 \leq i \leq j} t_i \Big) - T_j \quad \text{for} \quad 2 \leq j \leq n.$$

As in the proof of Proposition 2.5 the T_j are the intermediate results and the F_j are the errors accumulated during the first j multiplications. Note again that by assumption no overflows occur during the computations of the T_j. For $3 \leq j \leq n$ we have:

$$\begin{aligned} |F_j| &= \Big| \underbrace{\Big(\prod_{1 \leq i \leq j-1} t_i \Big)}_{F_{j-1}+T_{j-1}} t_j - (T_{j-1} \otimes t_j) \Big| \\ &= |F_{j-1} t_j + T_{j-1} t_j - (T_{j-1} \otimes t_j)| \\ &\leq |F_{j-1} t_j| + |T_{j-1} t_j - \underbrace{(T_{j-1} \otimes t_j)}_{T_j}| \\ &\leq |F_{j-1} t_j| + 2^{l(T_j)-M} \\ &= |F_{j-1}||t_j| + 2^{l(T_j)-M}, \end{aligned}$$

where we used again part 2 of Assumption (#) for the last inequality.

By induction on $j \in \{2, \ldots, n\}$ we show

$$|F_j| \leq 2^{l(T_j)-M+2(j-2)},$$

which proves the assertion for $j = n$ as in the last proof. From part 4 of Assumption (#) we obtain

$$l(T_{j-1}) + l(t_j) \leq l(T_j) + 1.$$

$\underline{j = 2}$: Part 2 of Assumption (#).
$\underline{j > 2}$:

$$\begin{aligned}
|F_j| &\leq |F_{j-1}| \underbrace{|t_j|}_{\leq 2^{l(t_j)}} + 2^{l(T_j)-M} \\
&\leq 2^{l(T_{j-1}-M+2((j-1)-2)} 2^{l(t_j)} + 2^{l(T_j)-M} \\
&= 2^{\overbrace{l(T_{j-1})+l(t_j)}^{\leq l(T_j)+1}-M+2j-6} + 2^{l(T_j)-M} \\
&\leq 2^{l(T_j)-M+2j-5} + 2^{l(T_j)-M+\overbrace{2j-5}^{\geq 0}} \\
&= 2^{l(T_j)-M+2(j-2)}
\end{aligned}$$

□

Now we can estimate the error accumulating during the addition of several numbers of the same sign as well as the error accumulating during the multiplication of several numbers by an expression which only depends on $l(\alpha)$ (if α is the result of the computation). We need an estimation of the accumulated errors when computing the products

$$\pi_i := \otimes_{1 \leq j \leq \gamma_i} t_{ij}, \quad 1 \leq i \leq r$$

(which have all the same sign), and adding the products π_i afterwards. To obtain such an estimation by use of Propositions 2.5 and 2.6 we need two auxiliary results. These results express the following: When we multiply several numbers, the sign of the result will be correct, and the same assertion holds for the addition of several numbers of the same sign.

Lemma 2.7 *Let $t_1, \ldots, t_n \in \mathbf{D}$, $\alpha := \otimes_{1 \leq i \leq n} t_i$. If no overflow occurs during the computation we have that*

$$\alpha > (<, =)\ 0 \qquad \textit{implies} \qquad \prod_{1 \leq i \leq n} t_i > (<, =)\ 0,$$

if $M \geq 2n - 2$ holds.

Proof: In the case $\alpha = 0$ it follows from Proposition 2.6 (recall $n > 0$, which leads to $M > 2(n-2)$)

$$\left| \prod_{1 \leq i \leq n} t_i - 0 \right| \leq 2^{1-M+2(n-2)} < 1,$$

hence because all t_i are integer

$$\prod_{1\leq i\leq n} t_i = 0$$

holds. Now let $\alpha \neq 0$. Assume the assertion does not hold. Then we have

$$\Big|\prod_{1\leq i\leq n} t_i - \alpha\Big| \geq |\alpha|,$$

which leads to the following contradiction by Proposition 2.6:

$$\begin{aligned} |\alpha| &> 2^{l(\alpha)-2} \\ &\geq 2^{\overbrace{l(\alpha)-2-(M-2n+2)}^{l(\alpha)-M+2(n-2)}} \\ &\geq \Big|\prod_{1\leq i\leq n} t_i - \alpha\Big| \\ &\geq |\alpha| \end{aligned}$$

□

Lemma 2.8 *Let $t_1, \ldots, t_n \in \mathbf{D}$, all $t_i \geq 0$ or all $t_i \leq 0$. Let $\alpha := \oplus_{1\leq i\leq n} t_i$. If no overflow occurs during the computation of α we have*

$$\alpha > (<, =)\ 0 \qquad \Rightarrow \qquad \sum_{1\leq i\leq n} t_i > (<, =)\ 0,$$

if $M \geq 2n-2$ holds.

Proof: Analogously to the proof of Lemma 2.7, here we use Proposition 2.5. □

Now we are able to give the needed estimation. The following Proposition 2.9 is the basis of our method.

Proposition 2.9 *Let $t_{ij} \in \mathbf{D}$ $(1 \leq i \leq n,\ 1 \leq j \leq 4, n \geq 2)$, and let*

$$\alpha := \oplus_{1\leq i\leq n}(\otimes_{1\leq j\leq 4} t_{ij}).$$

We demand that all $\otimes_{1\leq j\leq 4} t_{ij}$ $(1 \leq i \leq n)$ are nonnegative or all nonpositive. Furthermore, let $M \geq 2n-2$. If no overflow occurs during the computation of α we have

$$\Big|\sum_{1\leq i\leq n}\prod_{1\leq j\leq 4} t_{ij} - \alpha\Big| \leq 2^{l(\alpha)-M+n+4}.$$

Proof: For $1 \leq i \leq n$ we define

$$a_i := \prod_{1\leq j\leq 4} t_{ij} \quad \text{and} \quad A_i := \otimes_{1\leq j\leq 4} t_{ij}.$$

By Proposition 2.6 we have for $1 \le i \le n$

$$|a_i - A_i| \le 2^{l(A_i)-M+4},$$

and from Proposition 2.5 we obtain (remember that all $A_i \ge 0$ or all $A_i \le 0$)

$$\Big| \sum_{1 \le i \le n} A_i - \underbrace{\oplus_{1 \le i \le n} A_i}_{\alpha} \Big| \le 2^{l(\alpha)-M+n-2}.$$

By Lemma 2.4 and part 3 of Assumption (#) it follows inductively that

$$l(A_i) \le l(\alpha) \quad \text{for} \quad 1 \le i \le n.$$

Due to Lemma 2.7 we have either all $a_i \ge 0$ or all $a_i \le 0$ if all $A_i \ge 0$ or all $A_i \le 0$, respectively. Hence we obtain the following estimation:

$$\begin{aligned}
\Big| \sum_{1 \le i \le n} a_i - \sum_{1 \le i \le n} A_i \Big| &= \Big| \sum_{1 \le i \le n} (|a_i| - |A_i|) \Big| \\
&\le \sum_{1 \le i \le n} |a_i - A_i| \\
&\le \sum_{1 \le i \le n} 2^{\overbrace{l(A_i)}^{\le l(\alpha)}-M+4} \\
&\le n 2^{l(\alpha)-M+4} \\
&\le 2^{l(\alpha)-M+n+3},
\end{aligned}$$

where the last estimation holds since $n \le 2^{n-1}$ (we have $n > 1$).

Now we can estimate the error as follows:

$$\begin{aligned}
\Big| \sum_{1 \le i \le n} \prod_{1 \le j \le 4} t_{ij} - \alpha \Big| &= \Big| \sum_{1 \le i \le n} a_i - \oplus_{1 \le i \le n} A_i \Big| \\
&= \Big| \sum_{1 \le i \le n} a_i - \sum_{1 \le i \le n} A_i + \sum_{1 \le i \le n} A_i - \oplus_{1 \le i \le n} A_i \Big| \\
&\le \Big| \sum_{1 \le i \le n} a_i - \sum_{1 \le i \le n} A_i \Big| + \Big| \sum_{1 \le i \le n} A_i - \oplus_{1 \le i \le n} A_i \Big| \\
&\le 2^{l(\alpha)-M+n+3} + 2^{l(\alpha)-M+\overbrace{n-2}^{\le n+3}} \\
&\le 2^{l(\alpha)-M+n+4}
\end{aligned}$$

□

Now we know (by using the same terminology as before) an estimation for

$$\Big| P - P_{ex} \Big| \quad \text{and} \quad \Big| |N| - |N_{ex}| \Big|.$$

We derive a test that guarantees that $P_{ex} - |N_{ex}|$ has the same sign as $P - |N|$. For the moment, we assume $P < |N|$. Let F_P resp. F_N be the estimations of the errors $|P - P_{ex}|$

and $||N| - |N_{ex}||$. Let $P' := P + F_P$ and $N" := |N| - F_N$. Then the required test is visualized in Figure 1.

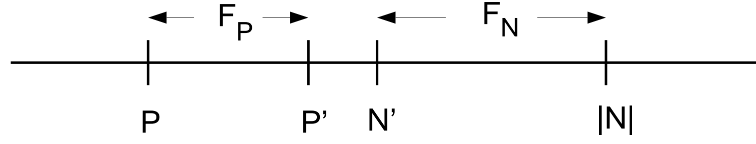


Figure 1: Illustration of the required test

If we have $P' < N'$ then the test is passed, otherwise we have to apply the software arithmetic. Hence we are searching (in the case of $P < |N|$) P' and N', which can be computed from P resp. N, so that

$$P' \geq P_{ex} \quad \text{and} \quad N' \leq |N_{ex}|.$$

Proposition 2.11 will show how to determine P' and N'. In order to prove this proposition we need the following lemma.

Lemma 2.10 *Let $a, b \in \mathbf{D}$ and $\odot \in \{\oplus, \ominus\}$. Then*

$$l(a \odot b) \leq max\{l(a), l(b)\} + 2$$

holds if no overflow occurs during the computation.

Proof: Let $\bullet \in \{+, -\}$ be the ring operation corresponding to $\odot$. We define

$$\lambda := \max\{l(a), l(b)\}.$$

Assume that $l(a \odot b) > \lambda + 2$. We know that $l(a \bullet b) \leq \lambda + 1$ holds. From the assumption above it follows that $2|a \bullet b| < |a \odot b|$. Then we obtain by part 2 of Assumption (#) the following contradiction (remember that $M > 8$):

$$\begin{aligned} 2^{l(a \odot b) - M} &\geq \Big|(a \odot b) - (a \bullet b)\Big| \\ &\geq \Big||a \odot b| - |a \bullet b|\Big| \\ &= |a \odot b| - |a \bullet b| \\ &> |a \odot b| - \tfrac{1}{2}|a \odot b| \\ &= \tfrac{1}{2}|a \odot b| \end{aligned}$$

□

Proposition 2.11 *Let $t_{ij} \in \mathbf{D}$ $(1 \leq i \leq n,\ 1 \leq j \leq 4, n \geq 2)$ and define*

$$\alpha := \oplus_{1 \leq i \leq n}(\otimes_{1 \leq j \leq 4} t_{ij}).$$

We demand that all $\otimes_{1 \leq j \leq 4} t_{ij}$ $(1 \leq i \leq n)$ are nonnegative or all nonpositive, that no overflow occurs and that we have $M \geq 2n - 2$. Then with

$$F := \lfloor 2^{l(\alpha) - M + n + 5} \rfloor \quad (\in \mathbf{D})$$

the following two statements hold.

(i) $\alpha \oplus F \geq \sum_{1\leq i\leq n} \prod_{1\leq j\leq 4} t_{ij}$

(ii) $\alpha \ominus F \leq \sum_{1\leq i\leq n} \prod_{1\leq j\leq 4} t_{ij}$

Proof: We only show *(i)*. The proof of *(ii)* is completely analogous. We define

$$\beta := \alpha \oplus F.$$

Since $M > 8$ we have

$$F < 2^{l(\alpha)+n-3},$$

which leads to

$$l(F) \leq l(\alpha) + n - 3.$$

Of course, then we also can give the following upper bounds on $l(F)$ and $l(\alpha)$:

$$l(F), l(\alpha) \leq l(\alpha) + n + 2,$$

which implies

$$l(\beta) \leq l(\alpha) + n + 4$$

by Lemma 2.10.

If $F = 0$ then $l(\alpha) < M - (n + 5)$ must hold, which implies

$$l(\beta) \leq l(\alpha) + n + 4 < M - 1,$$

from which in this case (since β is integer by part 1 of Assumption (#)) follows that

$$\alpha \oplus F = \alpha + 0 = \alpha.$$

From $F = 0$ we get by Proposition 2.9 that α was computed correctly. Hence in this case we even have equality in *(i)*.

If $F \neq 0$ then we have

$$F = 2^{l(\alpha)-M+n+5}.$$

Due to part 2 of Assumption (#) and Proposition 2.9 the following holds:

$$\begin{aligned} \beta &\geq \alpha + F - 2^{l(\beta)-M} \\ &= \underbrace{\alpha + 2^{l(\alpha)-M+n+4}}_{\geq \sum_{1\leq i\leq n}\prod_{1\leq j\leq 4} t_{ij}} + 2^{l(\alpha)-M+n+4} - \underbrace{2^{l(\beta)-M}}_{\leq 2^{l(\alpha)+n+4-M}} \\ &\geq \sum_{1\leq i\leq n} \prod_{1\leq j\leq 4} t_{ij} \quad , \end{aligned}$$

□

Now we can describe our method. By Proposition 1.1 we have to determine signs of expressions of the form

$$\sum_{1\leq i\leq r} \left(s_i \prod_{1\leq j\leq \gamma_i} t_{ij} \right),$$

where the t_{ij} are certain coordinate differences and $s_i \in \{-1, +1\}$. Obviously we can restrict $\gamma_i = 4$ for all i by the introduction of additional factors $(1 - 0)$. This simplifies the analyses. First we make the following further

Assumptions:

1. The (nonnegative integer) coordinates of the generators have to be less than 2^{M-3}.
2. The coordinate differences have to be bounded by a $C \in \mathbb{N}$ with $6C^4 < 2^L$.

The first assumption guarantees in combination with Lemma 2.10 and part 2 of Assumption (#) that all the coordinate differences t_{ij} (which are integer by part 1 of Assumption (#)) are computed exactly.
The second assumption guarantees that during all the computations no overflows occur. This holds since $6C^4$ is a bound for the absolute values of all result of computations in Proposition 1.1, if C is a bound for all the coordinate differences.

Remark 2.12 *If the floating point arithmetic of the used system works according to the* IEEE STANDARD *[4] with the DOUBLE format then we have seen in remark 2.3 that we can choose $M = 53$ and $L = 1024$. If all coordinates are in the set $\{0, \ldots, 2^{50} - 1\}$, then both assumptions are satisfied when using this format with $C := 2^{50}$ (e.g., on the machine we used to obtain our computational results).*

For each of the three functions `Closer`, `CheckVP` and `OnLeftSide` we specify the number of additions r and the corresponding sign s_i. When any of these three functions is called, we determine coordinate differences t_{ij}. Up to now no numerical error has arisen (under the assumptions made before). After that we have to evaluate only the sign of the expression

$$\sum_{1 \le i \le r} \left(s_i \prod_{1 \le j \le 4} t_{ij} \right).$$

To perform this evaluation we use the following routine SIGNOFEXPRESSION. As described in the beginning of this section we first determine P and N. Then by using the criterion obtained from Proposition 2.11, we check if we can guarantee that $P - |N|$ has the same sign as $P_{ex} - |N_{ex}|$. According to the result of this test either the sign of the expression is taken immediately as that of $P - |N|$ or the software arithmetic is called.

Routine SIGNOFEXPRESSION

INPUT : $s_i \in \{-1, +1\}, t_{ij} \in \mathbf{D}$ for $1 \le i \le r$ and $1 \le j \le 4$
OUTPUT : Sign (PLUS, MINUS, ZERO) of $\sum_{1 \le i \le r} (s_i \prod_{1 \le j \le 4} t_{ij})$

(1) For $1 \leq i \leq r$ compute the $T_i := s_i \otimes (\otimes_{1\leq j\leq 4} t_{ij})$
(2) Let w.l.o.g. $T_1, \ldots, T_m > 0$ and $T_{m+1}, \ldots, T_{m'} < 0$.
(3) Compute
$P := \oplus_{1\leq i\leq m} T_i,$
$N := \oplus_{m+1\leq i\leq m'} T_i$
(4) Define
$B_P := abs(P)$
$B_N := abs(N)$
(5) if $B_P \geq B_N$ then
{
(6) if $B_P > B_N$ then
(7) $tempres :=$ PLUS
(8) else
(9) $tempres :=$ ZERO
(10) $\alpha_{big} := B_P$
(11) $n_{big} := m$
(12) $\alpha_{small} := B_N$
(13) $n_{small} := m' - m$
}
(14) else
{
(15) $tempres :=$ MINUS
(16) $\alpha_{big} := B_N$
(17) $n_{big} := m' - m$
(18) $\alpha_{small} := B_P$
(19) $n_{small} := m$
}
(20) Determine
$F_{big} := \lfloor 2^{l(\alpha_{big})-M+n_{big}+5} \rfloor$
$F_{small} := \lfloor 2^{l(\alpha_{small})-M+n_{small}+5} \rfloor$
(21) Compute
$\beta_{big} := \alpha_{big} \ominus F_{big}$
$\beta_{small} := \alpha_{small} \oplus F_{small}$
(22) if $tempres =$ ZERO and ($F_{big} \neq 0$ or $F_{small} \neq 0$) then goto (failed)
(23) if $tempres \neq$ ZERO and $\beta_{big} \leq \beta_{small}$ then goto (failed)
(24) return $tempres$

(failed) Do the whole computation with software arithmetic
and return the obtained result.

To prove the correctness of the routine SignOfExpression we need a trivial auxiliary result that we formulate in the following lemma.

Lemma 2.13 *Let $t_1, \ldots, t_n \in \mathbf{D}$. Then*

$$|\otimes_{1\le i\le n}\, t_i| = \otimes_{1\le i\le n}|t_i|$$

holds. If furthermore all $t_i \ge 0$ or all $t_i \le 0$ then we also have

$$|\oplus_{1\le i\le n}\, t_i| = \oplus_{1\le i\le n}|t_i|.$$

Proof: The assertions follow (in the second case due to Lemma 2.4) inductively from the parts 8 and 7 of Assumption (#). □

Proposition 2.14 *The routine* SIGNOFEXPRESSION *works correctly.*

Proof: Under the assumption that the software arithmetic works correctly we only have to show that *tempres* contains the the sign of the expression which we want to evaluate when line (24) is reached.

We need two observations:

Due to Lemma 2.13 by construction of B_P and B_N we have

$$B_P = \oplus_{1\le i\le m}(\otimes_{1\le j\le 4}|t_{ij}|) \quad \text{and} \quad B_N = \oplus_{m+1\le i\le m'}(\otimes_{1\le j\le 4}|t_{ij}|) \quad (1).$$

Furthermore it follows by Lemma 2.7 that

$$\sum_{1\le i\le r}\left(s_i \prod_{1\le j\le 4} t_{ij}\right) = \sum_{1\le i\le m}\prod_{1\le j\le 4}|t_{ij}| - \sum_{m+1\le i\le m'}\prod_{1\le j\le 4}|t_{ij}| \quad (2).$$

Case 1: *tempres* = ZERO

Because of line (22) we have $F_{big} = F_{small} = 0$, so by Proposition 2.11 α_{big} and α_{small} (i.e., B_P and B_N) are computed correctly. On the other hand, since *tempres* = ZERO we have $B_P = B_N$ (because of line (9)), which by (1) and (2) leads to

$$\sum_{1\le i\le r}\left(s_i \prod_{1\le j\le 4} t_{ij}\right) = 0.$$

Case 2: *tempres* $\neq$ ZERO

Because of line (23) we have $\beta_{big} > \beta_{small}$. We only prove the case *tempres* = PLUS, the other case can be shown completely analogously.

If *tempres* = PLUS then we have (because of line (7)) $B_P > B_N$, hence $\alpha_{big} = B_P$ and $\alpha_{small} = B_N$. Due to the definition of β_{big} and β_{small} (because of line (21)) it follows by Proposition 2.11 that

$$\sum_{1\le i\le m}\prod_{1\le j\le 4}|t_{ij}| \ge \beta_{big} > \beta_{small} \ge \sum_{m+1\le i\le m'}\prod_{1\le j\le 4}|t_{ij}|,$$

and we obtain by (2)

$$\sum_{1\le i\le r}\left(s_i \prod_{1\le j\le 4} t_{ij}\right) > 0.$$

□

3 Computational results

We implemented our method on a SUN SPARCstation 10 model 41 using the programming language C. We chose SUN's compiler `acc (1.0)` with the compiler optimization option `-fast`. The software arithmetic we used for the experiments reported below is the package `GNU MP (1.3.2)` of the *Free Software Foundation, Inc.*. By sending an e-mail request to `kaibel@informatik.uni-koeln.de` you can obtain the library `delvor` (together with a short documentation) that provides a function to compute Delaunay triangulations (for the Euclidean metric as well as for Manhattan metric and Maximum metric, cf. JÜNGER, KAIBEL AND THIENEL [6] and KAIBEL [7]) and another function to calculate the corresponding Voronoi diagram from any Delaunay triangulation obtained by the first one.

The Euclidean part of our program can be run in three different modes. These modes determine how the computation of the functions `Closer`, `CheckVP` and `OnLeftSide` (see Section 1) are performed. The options are

Soft:	All computations are done by the software arithmetic.
Double:	All computations are done by the (double) floating point arithmetic.
SoX:	All computations are done by the routine SIGNOFEXPRESSION.

The entries in the columns titled by one of these three names are the numbers of software arithmetic calls (if these entries are integral) or the running time (if they are decimal fractions).

There is a column titled *OK* in every table. This column indicates if the output produced by the computation with option *Double* is different from the one obtained when using the option *Soft*. A difference is indicated by a $\bullet$, consistent results by a $\surd$. Finally, we have a column *Max* that indicates the maximum size of the generator coordinates of the problem. The problems in Table 3 are real world problems and have floating point coordinates. As mentioned in Section 1 we scale the coordinates to integer values before starting the computation. In such a case *Max* is the maximum size of the coordinates after the scaling.

We give three tables describing computational results. The first treats random problems (i.e., the generators are pseudo uniformly distributed on a nonnegative grid) where the number of the generators is varied, while the maximal size of the coordinates of the generators is fixed to 10^7. In the second, we vary the maximal size of the generator coordinates, while the number of generators is fixed to 10^4. The third contains some data for computations of Delaunay triangulations of problems given in the TSPLIB by REINELT [10].

As one can see in Table 3, there are indeed problems which cannot be computed correctly (by our program) when using the hardware floating point arithmetic. However, we have to mention that we do not know any problem that makes our program trap into an infinite loop when using the option *Double*. But surely one can construct such a problem.

All the tables show that the criterion developed in Section 2 reduces the number of necessary software arithmetic calls enormously. Actually, among our tested random problems in no case such a call was necessary.

Comparing the running times of the different methods one observes that the software arithmetic leads to a running time increased by factor between about 4 and 5, if this arithmetic is used for all computations (in comparison to the method which only uses double arithmetic). One sees that this factor can be reduced to a factor about 2 by using the routine SignOfExpression.

Table 1: Random problems with various numbers of generators

#gen	Soft	SoX	Double	Soft	SoX	ok
4	48	0	0.00	0.00	0.00	√
8	113	0	0.00	0.02	0.00	√
16	256	0	0.00	0.02	0.00	√
32	572	0	0.00	0.03	0.00	√
64	1253	0	0.02	0.08	0.03	√
128	2598	0	0.02	0.17	0.05	√
256	5207	0	0.07	0.35	0.10	√
512	11005	0	0.10	0.68	0.20	√
1024	21915	0	0.22	1.37	0.40	√
2048	45670	0	0.45	2.85	0.82	√
4096	88144	0	0.87	5.63	1.63	√
8192	184443	0	1.88	11.15	3.55	√
16384	355172	0	3.98	21.38	7.25	√
32768	740284	0	8.20	42.91	14.95	√
65536	1417039	0	16.78	81.75	30.70	√
131072	2956071	0	33.92	171.64	62.70	√

Table 2: Random problems with various ranges of generator coordinates

Max	Soft	SoX	Double	Soft	SoX	ok
100	169430	0	1.68	7.67	2.70	√
1000	227383	0	2.30	11.73	4.32	√
10000	228246	0	2.27	12.05	4.45	√
99993	228391	0	2.28	12.33	4.37	√
999929	228339	0	2.33	12.25	4.43	√
9999281	228345	0	2.30	13.80	4.45	√

Table 3: Problems out of the TSPLIB

Problem	Max	Soft	SoX	Double	Soft	SoX	ok
rd100	16088597	1956	0	0.02	0.13	0.05	√
pr124	13586	2583	0	0.02	0.13	0.03	√
bier127	5047	2807	0	0.03	0.13	0.05	√
pr152	15826	3244	0	0.03	0.17	0.05	√
d198	16499918	4455	26	0.05	0.23	0.08	•
gil262	199	5399	0	0.05	0.27	0.10	√
lin318	4135	6661	0	0.07	0.33	0.12	√
rd400	16343336	8534	0	0.08	0.53	0.17	√
pr439	13701	9822	0	0.10	0.50	0.17	√
pcb442	3801	9491	0	0.08	0.45	0.17	√
d657	16406939	15158	7	0.13	1.00	0.28	•
dsj1000	1214610	25405	0	0.22	1.22	0.42	√
u1060	10914305	23470	16	0.23	1.87	0.50	√
vm1084	37857	23520	0	0.27	1.10	0.38	√
pcb1173	3453	25497	0	0.27	1.38	0.50	√
d1291	16225076	29767	172	0.27	1.45	0.45	√
rl1323	19089	29580	0	0.28	1.43	0.50	√
nrw1379	8175	30971	0	0.33	1.65	0.62	√
fl1400	8620484	38644	84	0.30	1.70	0.57	√
u1432	2151	31987	0	0.33	1.75	0.55	√
fl1577	8559945	36763	128	0.33	1.87	0.60	√
d1655	15262516	38715	162	0.35	2.00	0.62	•
vm1748	38133	40774	2	0.38	1.95	0.68	√
u1817	13903422	42136	144	0.50	2.22	0.63	√
d2103	8905115	51238	257	0.50	2.58	0.82	•
u2152	14132880	50775	167	0.47	2.53	0.85	•
u2319	2151	54747	0	0.57	2.57	0.97	√
pr2392	15976	53384	0	0.52	2.78	0.93	√
pcb3038	3951	69209	0	0.65	3.48	1.25	√
fl3795	8620484	107061	150	0.95	4.75	1.57	•
fnl4461	10676	99852	0	0.95	5.22	1.85	√
rl5934	19153	135167	0	1.38	6.57	2.27	√
d6960	6077	164223	0	1.63	7.77	2.75	√
pla7397	627926	179208	3463	1.77	8.82	3.17	√
rl11849	38305	289533	1	2.88	13.57	4.85	√
brd14051	10967	350319	0	3.33	17.65	6.43	√
d18512	10967	419109	0	4.33	21.58	7.92	√
pla33810	697901	782286	12914	8.03	38.05	13.67	√
pla85900	726001	1956078	15186	20.28	92.96	33.15	√

References

[1] S. Fortune and C. J. van Wyck: Efficient Exact Arithmetic for Computational Geometry, Proceedings of the 9th Annual ACM Symposium on Computational Geometry (1993), 163-172.

[2] S. Fortune: A Sweepline Algorithm for Voronoi Diagrams, *Algorithmica* (1987) 2, 153-174.

[3] P. J. Green and R. Sibson: Computing Dirichlet Tesselations in the Plane, *Comput. J. 21* (1977), 168-173.

[4] IEEE Standard for Binary Floating-Point Arithmetic, *ANSI/IEEE-Std. 754-1985.*

[5] M. Jünger, G. Reinelt and D. Zepf: Computing Correct Delaunay Triangulations, *Computing* 47 (1991), 43-49.

[6] M. Jünger, V. Kaibel and S. Thienel: Computing Delaunay-Triangulations in Manhattan and Maximum Metric. (in preparation)

[7] V. Kaibel: Delaunay-Triangulation in verschiedenen Metriken, Diploma thesis, Institut für Informatik, Universität zu Köln (1993).

[8] T. Ohya, M. Iri und K. Murota: Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparisons of Various Algorithms, *Journal of the Operations Research Society of Japan* 27 (1984), 306-337.

[9] M. I. Shamos und D. Hoey: Closest-Point Problems, *Proceedings of the 16th Annual IEEE Symposium on FOCS* (1975), 151-162.

[10] G. Reinelt: TSPLIB – A Traveling Salesman Problem Library, *ORSA Journal on Computing 3* (1991), 376-384.

[11] K. Sugihara: A Simple Method for Avoiding Numerical Errors and Degeneracy in Voronoi Diagram Construction, *Research Memorandum RMI 88-14, Faculty of Engineering, University of Tokyo* (1988).

[12] K. Sugihara und M. Iri: Geometric Algorithms in Finite-Precision Arithmetic, *Research Memorandum RMI 88-10, Faculty of Engineering, University of Tokyo* (1988).

Michael Jünger
Institut für Informatik
Universität zu Köln
Pohligstr. 1
D-50969 Köln
Germany
Telephone: 49 221 4705313
e-mail: mjuenger@informatik.uni-koeln.de

Volker Kaibel
Institut für Informatik
Universität zu Köln
Pohligstr. 1
D-50969 Köln
Germany
Telephone: 49 221 4705314
e-mail: kaibel@informatik.uni-koeln.de

Stefan Thienel
Institut für Informatik
Universität zu Köln
Pohligstr. 1
D-50969 Köln
Germany
Telephone: 49 221 4705307
e-mail: thienel@informatik.uni-koeln.de